

Binary Context: A Practical Way to Think About Neural Networks

Much of deep learning can be understood in surprisingly simple terms.

Rather than thinking about neurons "computing features," it is often more useful to think of them as selecting **which computation should happen next**.

The selected computation is almost always linear.

The interesting part is not the linear mapping itself. The interesting part is **how the network chooses which linear mapping to use**.

Binary context

Consider a layer with a binary context vector.

The context might come from:

- ReLU activations (positive or zero)
- threshold units
- locality sensitive hash (LSH) bits
- routing networks
- mixture-of-experts gates
- any other binary routing mechanism

The binary vector does not directly solve the problem.

Instead it answers a much simpler question:

Which parameters should participate in the next computation?

Once those parameters are selected, the layer performs an ordinary linear mapping.

So every layer is essentially

Context → parameter selection → linear mapping

Backpropagation learns the average mapping

Suppose one particular context occurs many thousands of times during training.

Those examples are not identical.

They are simply similar enough that they activate the same context.

Backpropagation gradually adjusts the selected parameters to perform as well as possible over all those examples.

The result is not a perfect mapping for every sample.

It is a **best-effort average linear mapping** for that context.

This viewpoint explains why the same parameters can successfully process many different inputs.

The parameters are not memorizing individual examples.

They are learning the linear transformation that works best on average for everything routed through that context.

Every layer repeats the process

Each layer performs exactly the same operation.

1. Determine the current context.
2. Select a subset of parameters.
3. Apply the corresponding linear mapping.
4. Pass the result to the next layer.

The overall network becomes a composition of many context-dependent linear mappings.

Rather than learning one gigantic nonlinear function directly, the network stitches together many locally useful linear approximations.

ReLU networks already work this way

This viewpoint is not limited to Atlas networks.

Even an ordinary ReLU network behaves similarly.

The ReLU pattern determines which neurons are active.

That activation pattern determines which columns and rows of the weight matrices actually contribute to the computation.

The resulting computation is linear for that activation pattern.

Training therefore learns many overlapping linear mappings corresponding to different activation patterns.

Atlas networks make the context explicit

Atlas networks simply move this idea one step further.

Instead of allowing every layer to generate its own local binary context, the network receives an explicit context generated from the input.

For example, an LSH projection may generate hundreds of binary routing bits.

Those bits select parameter subsets throughout the network.

Each layer therefore asks

Given this input context, what is the best linear mapping I can learn?

The layer does not need to discover the context.

It receives the context directly.

Its job is simply to optimize the selected parameters.

A computational atlas

The name "Atlas" comes from geography.

An atlas contains many maps.

Each map covers one region.

Similarly, the binary context identifies one region of computational space.

Each region owns a collection of parameters.

Those parameters learn the best computation for that region.

Training gradually fills in the atlas.

Some regions receive many examples and become highly refined.

Others remain coarse because they are rarely visited.

One of the surprising successes of backpropagation

Perhaps the most surprising practical observation is that backpropagation handles extremely fragmented parameter usage remarkably well.

During one training example, perhaps only a small fraction of the network's parameters are updated.

The next example may update a largely different subset.

At first this seems problematic.

How can coherent learning emerge if parameters are constantly skipped?

Experience suggests that gradient descent is surprisingly tolerant of this fragmented update pattern.

This is visible in

- ReLU networks
- sparse mixture-of-experts models
- conditional computation
- routing architectures
- sparse activation methods

Only part of the network participates for each example, yet useful global behaviour still emerges.

This robustness is one of the underappreciated strengths of stochastic gradient descent.

Why averaging works

No routing mechanism is perfect.

Every context inevitably contains variation.

The selected parameters therefore see a cloud of related examples rather than identical ones.

Backpropagation naturally learns the linear mapping that reduces error across that cloud.

This averaging process is exactly what gives each context useful generalization instead of simple memorization.

If the contexts become more discriminative, each cloud becomes tighter.

Tighter clouds are generally easier to approximate with a linear mapping.

This suggests an interesting design principle:

Better contexts can reduce the burden placed on the linear learner.

Context quality matters

The network's success depends heavily on the quality of its contexts.

Useful contexts should ideally

- group similar problems together
- separate genuinely different problems
- occur often enough to learn reliable parameters
- avoid pathological imbalance
- preserve enough diversity to generalize

In Atlas-style networks, much of the architectural design revolves around producing useful binary contexts.

Local specialization

Each context becomes a local expert.

Unlike classical mixture-of-experts systems, the expertise is not necessarily represented by an entire subnetwork.

It may simply be a different selection of parameters inside otherwise shared layers.

This allows specialization without requiring every parameter to be unique.

Composition creates nonlinearity

Each selected mapping is linear.

The network as a whole is not.

The nonlinear behaviour comes entirely from changing which linear mapping is selected as the context changes.

This is a powerful perspective.

Instead of viewing nonlinearity as something mysterious happening inside neurons, it can be viewed as the consequence of switching among many learned linear computations.

A practical engineering viewpoint

From an engineering perspective, each layer has two separate jobs.

The context mechanism

- decides where this input belongs.

The linear learner

- learns the best average mapping for everything assigned there.

Keeping these roles conceptually separate often simplifies architectural thinking.

An intuition

Imagine a workshop.

A dispatcher looks at each incoming job and decides which bench should handle it.

Each bench owns a different set of tools.

The workers at that bench become increasingly skilled because they repeatedly see similar jobs.

No bench performs perfectly on every possible job.

Each simply becomes very good at the jobs routed to it.

An Atlas neural network can be viewed similarly.

The binary context acts as the dispatcher.

The selected parameters are the specialised workbench.

Backpropagation continually improves each workbench using the jobs that arrive there.

The complete computation is then the sequence of specialised workbenches that the input visits as it passes through the network.

Closing thought

One of the simplest ways to view modern neural networks is this:

- Context determines which parameters are active.
- Active parameters define a linear mapping.
- Backpropagation learns the best average mapping for each context.
- Layers compose these context-dependent mappings into a powerful nonlinear computation.

Atlas-style networks take this viewpoint seriously by making context an explicit architectural component rather than an implicit consequence of neuron activations.

Whether or not Atlas architectures ultimately prove useful in practice, this decomposition—**context first, linear mapping second**—offers a practical way to reason about conditional computation, sparse learning, and the remarkable ability of gradient descent to train systems in which different examples continually activate different pieces of the model.